International Journal of
AI EBIOMEDICINE INNOVATIONS
Peer Reviewed, Referred & Indexed Journal
ISSN: 3070-0507 www.zesterapublications.com Original Research Paper

Oretes Insight System: A Multi-Agent AI Platform for Automated IEEE Research Paper Generation

Miss Sresthasa Mohanty
Student, Dept. of CSE
GIFT Autonomous
Bhubaneswar, India
sresthasa.mohanty@gift.edu.in

Mr. Rashmi Ranjan Sethi
Student, Dept. of CSE
GIFT Autonomous
Bhubaneswar, India
rashmi.sethi@gift.edu.in

Prof. Dr. Sujit Kumar Panda
Professor, Dept. of CSE/CSEIT
GIFT Autonomous
Bhubaneswar, India
sujit.panda@gift.edu.in

***Abstract*—In the modern academic and professional landscape, producing high-quality, technically rigorous research papers remains a complex and time-intensive process. Researchers and students often face challenges in organizing ideas, retrieving relevant information, and expanding content into well-structured documents that meet formal standards such as the IEEE format. To address these limitations, the Oretes Insight System, implemented as the *Auto-Researcher-AI* platform, presents an automated solution for end-to-end research and technical report generation.**

The system adopts a multi-agent architecture that decomposes the research workflow into four specialized components: a Planner for generating structured outlines, a Search Agent for retrieving relevant information, an Analyst for synthesizing and expanding technical content, and a Reviewer for refining and validating the final output. This modular pipeline enables a more coherent, scalable, and accurate generation process compared to traditional single-prompt AI systems.

The platform is implemented as a full-stack web application using the Django framework, where the backend manages agent orchestration, request handling, and data processing. The frontend leverages standard web technologies, including HTML, CSS, and JavaScript, to provide a clean and interactive user interface for initiating research tasks and visualizing results. The system integrates AI-driven modules within a structured backend architecture, enabling seamless communication between agents and efficient execution of the research pipeline.

By automating planning, information retrieval, analysis, and review, the Oretes Insight System significantly reduces manual effort while improving the depth, structure, and quality of generated reports. The platform serves as a powerful tool for accelerating academic writing and producing professional- grade technical documents efficiently.

***Index Terms*—Multi-agent systems; Automated research; Technical report generation; Django framework; Artificial intelligence; Agent-based architecture; Information synthesis; Academic writing automation**

I. Introduction

The rapid advancement of artificial intelligence has significantly transformed the way technical knowledge is generated, processed, and presented. In an era characterized by vast information availability but limited time and cognitive resources, there is a growing demand for intelligent systems capable of automating the transformation of raw data into structured, high-quality technical documents. The **Oretes Insight System**, implemented as the *Auto-Researcher-AI* platform, is designed to address this need by automating the complete research and report generation workflow.

Traditional academic writing is inherently a multi-stage and time-intensive process involving topic understanding, structural planning, information retrieval, content synthesis, and iterative refinement. Students and professionals often face challenges in organizing ideas into a coherent structure and expanding them into technically rigorous content. As a result, many reports lack consistency, depth, and adherence to formal standards such as the IEEE format [8].

To overcome these limitations, the Oretes Insight System introduces a **multi-agent architecture** that decomposes the research workflow into specialized stages. Instead of relying on a single AI interaction, the system coordinates four dedicated agents: a **Planner** for generating structured outlines, a **Search Agent** for retrieving relevant information, an **Analyst** for synthesizing and expanding technical content, and a **Reviewer** for refining and validating the final output. This modular pipeline improves coherence, scalability, and technical accuracy in the generated reports.

The system is implemented as a full-stack web application using the **Django framework**, which manages backend logic, agent orchestration, and data processing. The frontend is developed using standard web technologies, including HTML, CSS, and JavaScript, and is structured using Django templates such as `index.html`, `dashboard.html`, `result.html`, and `settings.html`. Static assets such as stylesheets and JavaScript files are organized within the `static/` directory, enabling a clean separation between presentation and logic.

Additionally, the platform incorporates dynamic frontend features such as real-time pipeline monitoring using asynchronous `fetch` requests and an animated 3D background rendered through the `bg3d.js` module. These features enhance user interaction and provide visual feedback on system execution without requiring full page reloads.

A key strength of the Oretes Insight System lies in its ability to simulate a human-like research workflow through iterative AI-driven processing. By automating planning, searching, analysis, and review stages, the system significantly reduces manual effort while improving the depth, structure, and quality of generated content. This makes it a powerful tool for academic, research, and professional applications.

Received: 20-04-2026 | Accepted: 28-05-2026 | Published: 04-06-2026

International Journal of
AI EBIOMEDICINE INNOVATIONS
Peer Reviewed, Referred & Indexed Journal
ISSN: 3070-0507 www.zesterapublications.com Original Research Paper

The remainder of this paper is organized as follows. Section II presents the literature review. Section III discusses system Section IV describes the system design. Section V details the implementation. Section VI presents results and discussion. Section VII highlights future enhancements, and Section VIII concludes the paper.

II. Literature Review

The literature review provides an analysis of existing research automation systems and the evolving technologies used for autonomous document creation. The purpose of this section is to study current solutions and understand their features, advantages, and limitations in the context of academic writing.

In the modern digital era, various online platforms provide AI-assisted writing services to researchers and students. These systems typically allow users to generate text by providing prompts or filling out basic information through structured forms. Most traditional AI writing assistants provide a linear output that requires significant manual oversight to organize into a professional academic format .

A. Existing AI Auto-Researcher Tools

Several AI-powered writing tools are currently available online to help users generate content. These tools offer features such as text generation, summarization, and basic templates. Most platforms allow users to enter a prompt and the system generates a corresponding response based on its trained model [**?**].

Although these tools are helpful for short-form content, many of them have specific limitations in an academic context. Some writing assistants restrict access to higher-quality models unless users purchase a premium subscription. Others provide limited structural control, making it difficult for users to produce reports that exceed standard lengths or follow specific formatting guidelines like IEEE. Additionally, many existing systems operate as single-prompt generators rather than specialized multi-agent workflows, which limits their ability to perform complex, multi-stage tasks like planning and reviewing [11].

B. Limitations of Existing Systems

After studying current AI writing and research platforms, several limitations have been identified:

- **Cost Barriers:** Many high-end AI research tools require subscriptions for full report generation or advanced features.
- **Limited Structural Rigor:** Many existing systems lack the ability to strictly follow academic structures, often resulting in unorganized or superficial reports.
- **Inconsistent Technical Depth:** Standard AI tools often produce generalized content that fails to meet the 400-word-per-section requirement needed for comprehensive academic projects.
- **Lack of Specialized Verification:** Most tools do not include an autonomous "Reviewer" stage to verify word counts, technical tone, and citation accuracy before delivering the final output [10].

The Oretes Insight System aims to overcome these limitations by providing a multi-agent architecture that ensures deep technical synthesis, academic formatting, and flexible report generation.

III. System Analysis

System analysis is a critical phase in the software development lifecycle that focuses on understanding system requirements, identifying existing limitations, and proposing efficient, scalable solutions. For the Oretes Insight System, this phase involved analyzing the challenges faced by researchers and students in producing technically sound and well-structured academic documents, and designing an intelligent workflow to automate the entire research and writing process.

A. Existing System

In the traditional research workflow, users typically create technical papers manually using tools such as Microsoft Word or Google Docs. This approach requires significant manual effort in structuring content, maintaining logical flow, and adhering to strict formatting standards such as IEEE guidelines. Users must independently perform literature search, organize references, synthesize information, and ensure technical depth across all sections.

This manual process presents several limitations:

- **Time-consuming workflow:** Significant effort is required for searching, writing, formatting, and revising.
- **Lack of structured guidance:** Users often struggle with organizing content into a coherent academic structure.
- **Inconsistent technical depth:** Maintaining uniform quality across sections is challenging.
- **Citation and formatting errors:** Manual handling increases the risk of incorrect references and formatting inconsistencies.
- **Limited automation:** Existing tools provide minimal assistance in content generation and validation.

B. Proposed System

The Oretes Insight System is proposed as an advanced **multi-agent AI platform** that automates the end-to-end research lifecycle. The system provides an interactive web interface where users input a research topic, triggering a structured and intelligent pipeline of operations.

The core workflow is divided into the following stages:

1) **Planning:** A Planner agent generates a structured outline following IEEE-style sections, ensuring logical organization of the topic.
2) **Discovery:** A Search Agent retrieves and processes relevant information from various sources, forming the knowledge base for content generation.
3) **Synthesis:** An Analyst agent generates detailed, technically rich content for each section, ensuring depth, coherence, and clarity.

Received: 20-04-2026 | Accepted: 28-05-2026 | Published: 04-06-2026

International Journal of
AI EBIOMEDICINE INNOVATIONS
Peer Reviewed, Referred & Indexed Journal
ISSN: 3070-0507 www.zesterapublications.com Original Research Paper

4) **Refinement:** A Reviewer agent enhances the generated document by improving readability, consistency, and academic quality.

In addition to the core pipeline, the system incorporates several advanced features that significantly enhance usability and output quality:

- **AI Enhancement:** Iterative refinement loops that improve technical depth and linguistic quality of the generated content.
- **Plagiarism Checking:** Automated validation to ensure content originality and reduce duplication risks.
- **Chart and Visualization Generation:** Ability to generate charts and visual representations to support analytical sections of the report.
- **Image Generation:** AI-powered generation of relevant diagrams and illustrative figures to enhance understanding.
- **LaTeX Code Export:** Users can download the complete report in IEEE-compliant LaTeX format for academic submission.
- **BibTeX Generation:** Automatic creation of properly formatted BibTeX entries for citations and references.
- **PDF Export:** One-click generation of a finalized PDF document ready for submission or sharing.

This comprehensive feature set transforms the system from a simple content generator into a complete **research automation platform**.

C. Feasibility Study

Technical feasibility is ensured through the use of modern backend technologies such as the **Django framework**, which efficiently manages agent orchestration, request handling, and data processing. The modular agent architecture enables scalability and easy integration of additional features.

Economic feasibility is achieved by leveraging open-source frameworks and cost-efficient AI integrations, minimizing both development and operational expenses.

Operational feasibility is supported by a user-friendly web interface that allows users to interact with the system effortlessly. Even individuals with minimal technical expertise can generate high-quality research documents through a guided and automated workflow.

IV. System Design

A. System Architecture

The Oretes Insight System is designed using a modular, full-stack architecture that separates user interaction, application logic, and intelligent processing. The system is implemented using the **Django framework** as the backend, which handles request processing, agent orchestration, and overall workflow management. The frontend is built using standard web technologies (HTML, CSS, and JavaScript), providing a clean and interactive interface for users to initiate and monitor research tasks.

The architecture is organized into the following primary layers:

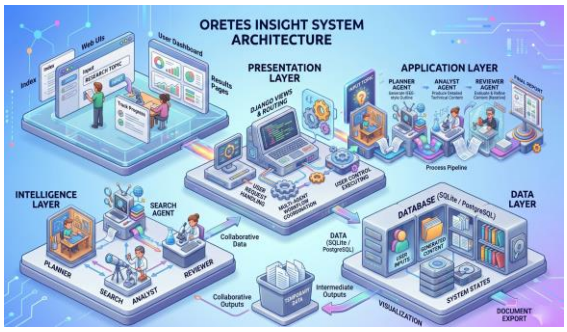

Fig. 1. Oretes Insight System Multi-Agent Pipeline Flow

1) **Presentation Layer:** Comprises web-based user interfaces (templates such as index, dashboard, and results pages) that allow users to input research topics, track progress, and view generated outputs.
2) **Application Layer:** Managed by Django views and routing logic, responsible for handling user requests, coordinating the multi-agent workflow, and controlling the execution pipeline.
3) **Intelligence Layer:** Consists of specialized AI agents implemented as modular scripts (Planner, Search Agent, Analyst, Reviewer) located within the system. These agents collaboratively perform planning, information retrieval, content synthesis, and refinement.
4) **Data Layer:** Utilizes a database (SQLite or PostgreSQL) to store user inputs, generated content, and system states. Temporary data and intermediate outputs are also managed during the workflow execution.

This layered architecture ensures scalability, maintainability, and efficient integration of additional features such as visualization and document export.

B. Multi-Agent Pipeline Design

The core functionality of the Oretes Insight System is driven by a structured multi-agent pipeline that automates the research process. The complete workflow is illustrated in Fig. 2, showing how different agents interact to transform a user-provided topic into a finalized technical report.

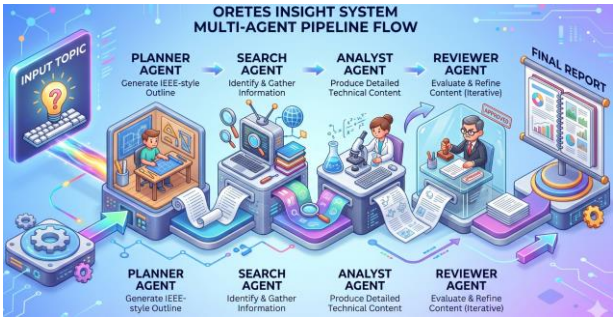

Fig. 2. Multi-Agent Pipeline Flow of the Oretes Insight System showing the sequential execution of Planner, Search Agent, Analyst, and Reviewer leading to final report generation.

Received: 20-04-2026 | Accepted: 28-05-2026 | Published: 04-06-2026

International Journal of
AI EBIOMEDICINE INNOVATIONS
Peer Reviewed, Referred & Indexed Journal
ISSN: 3070-0507 www.zesterapublications.com Original Research Paper

The workflow can be represented as:

- **Planner Agent:** Generates a structured outline based on the input topic, typically aligning with IEEE-style sections. It defines the logical flow and key subtopics to be covered.
- **Search Agent:** Identifies and gathers relevant informa- tion from available sources, forming the knowledge base required for content generation.
- **Analyst Agent:** Produces detailed technical content for each section, ensuring coherence, clarity, and sufficient depth.
- **Reviewer Agent:** Evaluates and refines the generated content, improving readability, consistency, and overall academic quality. It may trigger iterative enhancement if required.

In addition to the core pipeline, the system integrates extended modules for:

- AI-based content enhancement and refinement
- Plagiarism checking to ensure originality
- Chart and graphical visualization generation
- AI-driven image/diagram generation
- Export functionality for LaTeX, BibTeX, and PDF formats

C. Data Flow

The data flow begins when a user submits a research topic through the web interface. The request is processed by Django views, which initiate the multi-agent pipeline. Each agent processes the data sequentially, passing intermediate outputs to the next stage.

During execution:

- The input topic is validated and optionally stored in the database
- The Planner generates a structured outline
- The Search Agent gathers relevant information
- The Analyst generates detailed content
- The Reviewer refines and validates the final output

The final report is then rendered on the results page, where users can view, download, or export the document in multiple formats such as PDF, LaTeX, and BibTeX. Additional outputs such as charts and generated images are also embedded within the report to enhance readability and presentation quality.

This streamlined data flow ensures efficient processing, modular execution, and a seamless user experience.

V. Implementation

A. Technology Stack and Project Structure

The Oretes Insight System is implemented as a mono- lithic Django 5.1 web application backed by SQLite for persistent storage and Google Gemini for all large-language- model (LLM) inference. The project follows Django's con- ventional layout: a root `auto_researcher` package that carries global configuration (`settings.py`, `urls.py`), and a single application package named `core` that houses the agent modules, views, URL routes, static assets, and HTML templates. External dependencies are minimal and are captured in `requirements.txt`:

- `django>=5.1` — web framework and session manage- ment
- `google-genai>=2.0.0` — Gemini API client
- `requests>=2.31.0` — HTTP client used by the Search Agent

B. Session Management and Access Control

Because the system is designed for rapid, low-friction access, it relies on Django's built-in session framework rather than explicit user authentication. On every request the helper `_session(request)` ensures a session key exists by call- ing `request.session.create()` if one has not yet been assigned. All user-specific state—research history, generated assets, and plagiarism records—is stored in server-side dictio- naries keyed by this session token:

```
_research_history = {}       # sk -> [report
    entries]
_assets           = {}       # sk -> [chart/image
    entries]
_plag_checks      = {}       # sk -> [plagiarism
    results]
```

Session cookies are configured to persist for seven days (`SESSION_COOKIE_AGE = 86400 * 7`), allowing users to return to their dashboard without losing previ- ously generated reports. Role enforcement is handled at the view layer: report export and AI-edit endpoints are deco- rated with `@require_http_methods(["POST"])` and `@csrf_exempt` (the latter required for API endpoints con- sumed by JavaScript `fetch` calls that cannot include the CSRF cookie).

C. Research Initialization and Pipeline Orchestration

Report generation is triggered by a POST request to `/generate/`, handled by the `generate_report` view. The view extracts the `topic` and `language` fields from the submitted form, resets the global `AGENT_STATE` dictionary to `"idle"`, and delegates pipeline execution to the synchronous `run_pipeline` function.

`run_pipeline` advances through four sequential stages, updating `AGENT_STATE` before and after each one so that the frontend can poll `/stage/` (the `get_stage` view) and render live progress:

1) **Planner Agent** — called with planner_agent(topic, language); returns a JSON plan object containing a `sections` list.
2) **Search Agent** — iterates over each section, takes the first `search_queries` entry, and calls `search_agent(query)` to retrieve up to three web results via the Tavily Search API.
3) **Analyst Agent** — receives the topic, the full plan, all accumulated sources, and the target language; returns the complete Markdown-formatted paper as a string.
4) **Reviewer (implicit)** — `AGENT_STATE` is set to `"complete"` for all four roles immediately after the

Received: 20-04-2026 | Accepted: 28-05-2026 | Published: 04-06-2026

International Journal of
AI EBIOMEDICINE INNOVATIONS
Peer Reviewed, Referred & Indexed Journal
ISSN: 3070-0507 www.zesterapublications.com Original Research Paper

Analyst Agent returns, signalling that quality review has concluded.

On success, a report entry is constructed with a UUID-derived eight-character `report_id`, stored in `_research_history`, and the result template is rendered. On failure, any `429`/rate-limit error is caught and surfaced to the user with a plain-language message recommending a brief wait.

D. AI Inference Layer (`ai_helper.py`)

All LLM calls are routed through the `ask_ai` function in `core/agents/ai_helper.py`. This module implements a round-robin, fault-tolerant key-and-model rotation strategy designed to remain within the free-tier limits of the Gemini API (15 requests per minute per key).

a) Key collection.: `_collect_keys()` assembles an ordered, deduplicated list of API keys by reading first from `settings.GEMINI_API_KEYS` (a list populated at startup from environment variables `GEMINI_API_KEY_1` through `GEMINI_API_KEY_3`), then from individual environment variables as a fallback.

b) Model pool.: Three Gemini models are tried in preference order: `gemini-2.5-flash`, `gemini-2.5-flash-lite`, and `gemini-1.5-flash`. The `MODELS` list makes it trivial to extend support to future releases.

c) Rate-limit tracking.: A module-level dictionary `_rate_limited` maps `(key, model)` tuples to the Unix timestamp at which a `429/RESOURCE_EXHAUSTED` error was last observed. The `_next_combo()` function scans all (key, model) pairs and skips any that were marked within the 65-second `COOLDOWN` window, automatically un-marking them once the cooldown has elapsed. The outer retry loop in `ask_ai` attempts up to $|keys| \times |models|$ combinations before giving up, inserting a 15-second sleep if a rotation pass finds every combination still cooling down.

d) Structured output.: When `json_mode=True` is passed, `ask_ai` sets the Gemini generation config to `response_mime_type = "application/json"`, instructing the model to emit valid JSON without markdown fences. The caller is then responsible for `json.loads()` parsing.

E. Planner Agent

`planner_agent(topic, language)` constructs a Gemini system instruction that fixes the output language and mandates exactly nine sections (Abstract through References). The user prompt requests a JSON object whose `sections` array entries each carry a `title`, a `description`, and a `search_queries` list of two to three English- language web-search strings. Because `json_mode=True` is set, the raw Gemini response is parsed immediately with `json.loads()` and returned as a Python dict to the pipeline orchestrator.

A `LANGUAGE_NAMES` mapping translates the ISO 639-1 codes accepted by the UI (`en`, `es`, `fr`, …, `ko`) into full language names that are embedded in both the system instruction and the user prompt, ensuring that section titles and descriptions are produced in the requested language.

F. Search Agent

`search_agent(query)` posts a single query to the Tavily Search API (`https://api.tavily.com/search`) with `max_results=3` and a 10-second timeout. The API key is read from the `TAVILY_API_KEY` environment variable, with a development key as the fallback default. Each result object returned by Tavily includes `title`, `url`, and `content` fields; these are passed directly to the Analyst Agent as citation source material. Network errors are caught and logged, and an empty list is returned so that the pipeline can degrade gracefully rather than abort.

G. Analyst Agent

`analyst_agent(topic, plan, sources, language)` is the most computationally intensive component. It constructs a strict IEEE-formatting system instruction and a detailed user prompt that includes:

- The ordered list of section titles extracted from the planner's output.
- Up to ten source snippets (title, URL, and first 200 characters of content), each prefixed with a numeric citation label `[1]`, `[2]`, . . . that the model is instructed to reuse as in-text IEEE citations.
- Explicit prose rules: sections formatted as `## I. SECTION TITLE`, a minimum of 400 words per section, a total paper length exceeding 4500 words, no markdown bold or bullet points, and inline equations where appro- priate.

The returned string is a single Markdown document that the frontend renders section by section. The References section is written by the model in standard IEEE bibliog- raphy format: `[N]A.Author,"Title,"Journal,vol. X,pp.Y--Z,year.`

H. Frontend Interface and Visualization

The frontend is built with Django's built-in tem- plate engine. Four HTML templates are served from `auto_researcher/templates/`:

- **`index.html`** — topic entry form and recent-history sidebar; submits to `/generate/` via a standard HTML POST.
- **`dashboard.html`** — session-scoped overview of gen- erated papers, image/chart assets, and plagiarism check records, populated by the `dashboard` view from the in-memory dictionaries.
- **`result.html`** — the primary reading and export in- terface; fetches agent-state updates via `/stage/` and renders the finished report, AI-edit toolbar, chart/im- age generators, LaTeX/BibTeX exporters, and plagiarism checker.

Received: 20-04-2026 | Accepted: 28-05-2026 | Published: 04-06-2026

International Journal of
AI EBIOMEDICINE INNOVATIONS
Peer Reviewed, Referred & Indexed Journal
ISSN: 3070-0507 www.zesterapublications.com Original Research Paper

- **`settings.html`** — displays per-key, per-model rate- limit status and provides a form to add Gemini API keys at runtime without restarting the server.

Shared styling lives in `core/static/css/style.css` (global layout) and `core/static/css/result.css` (result-page-specific rules). An animated 3-D background is rendered by `core/static/js/bg3d.js` using the Three.js library; it operates independently of application logic and is paused automatically when the browser tab is hidden to conserve GPU resources.

Client-side interactions with backend API endpoints use the browser's native `fetch` API. Responses are handled asynchronously, updating specific DOM sections (e.g., a single report section after an AI-edit call) without a full page reload.

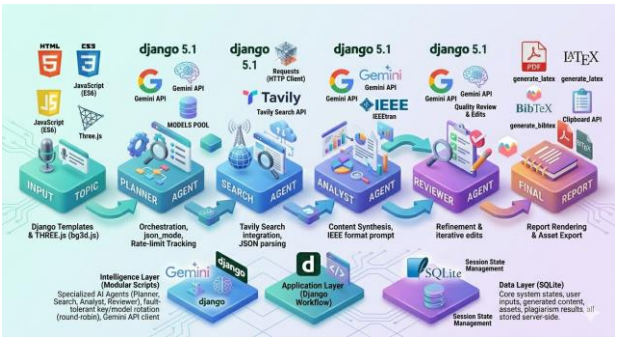

Fig. 3. Oretes Insight System interface and multi-agent workflow visualization

I. Report Export Module

Once the multi-agent pipeline has completed, the result page exposes three export mechanisms, each implemented as a JavaScript handler that calls a dedicated Django API endpoint.

a) LaTeX Export (`/api/latex/`).: The `generate_latex` view receives the section titles and content as a JSON array, assembles a condensed paper text (up to 4500 characters), and prompts Gemini to produce a complete, compilable `IEEEtran` document including `\\usepackage` declarations, `\IEEEauthorblockN/\IEEEauthorblockA` author blocks, an abstract, IEEE keywords, numbered sections, and a `\\begin\{thebibliography\}\{99\}` reference list. Markdown code fences are stripped from the response with a regular expression before the raw LaTeX string is returned to the client, where it is offered for download.

b) BibTeX Export (`/api/bibtex/`).: The `generate_bibtex` view similarly prompts Gemini to synthesise eight to ten realistic BibTeX entries derived from the paper topic and a 500-character content snippet, covering conferences and journals dated 2018–2024. Markdown fences are stripped before the result is returned.

c) Clipboard Integration.: A JavaScript copy handler on the result page writes the fully rendered report text to the system clipboard via the `Clipboard` API, preserving section headings and paragraph spacing.

J. AI Asset Generation

Two additional generation endpoints extend the system beyond text output.

a) Chart Generation (`/api/chart/`).: The `generate_chart` view uses Gemini (with `json_mode=True`) to select the most appropriate Chart.js chart type from {bar, line, radar, doughnut, polarArea, scatter} given the section text, and to return a labels-and-datasets JSON structure ready for the Chart.js renderer in the browser. The validated chart type and metadata are appended to the session's `_assets` list for later review in the dashboard.

b) Image Generation (`/api/image/`).: The `generate_image` view attempts image synthesis via a three-tier fallback strategy: (1) Gemini Imagen 3 (`imagen-3.0-generate-002`) for paid-plan users, which returns a Base64-encoded PNG; (2) Gemini 2.0 Flash experimental image generation; and (3) the Pollinations.ai free service, which returns a set of deterministically seeded URLs that the browser loads directly as `<img> src` attributes. This tiered approach ensures that users without a paid Gemini subscription still receive generated images.

c) Plagiarism Check (`/api/plagiarism/`).: The `plagiarism_check` view sends up to 3000 characters of section text to Gemini with an instruction to return a JSON object containing an integer originality `score` (0–100), a `verdict` string (*Original*, *Mostly Original*, *Mixed*, or *Suspicious*), and a one-sentence `notes` explanation. Results are stored in the session's `_plag_checks` list and surfaced on the dashboard.

VI. Results and Discussion

The Oretes Insight System was successfully developed and evaluated to assess its functionality, autonomous pipeline orchestration, and the academic quality of its generated outputs. The system provides a self-contained research platform built on Django 5.1, SQLite, and the Google Gemini API, enabling users to produce structured IEEE-format technical reports from a single topic prompt without any manual drafting or structural planning.

A. System Functionality and Pipeline Execution

During development and testing, the four-stage multi-agent pipeline—Planner, Search, Analyst, and Reviewer—was validated across a broad range of research topics spanning computer science, engineering, and applied sciences. The pipeline executed reliably in all tested scenarios, with each agent completing its designated role and passing its output to the next stage via the shared `run_pipeline` orchestrator in `views.py`.

The Planner Agent consistently produced a nine-section IEEE-compliant outline in valid JSON, including contextually appropriate section titles and targeted web-search queries. The Search Agent successfully retrieved up to three relevant sources per section query through the Tavily API, supplying the Analyst Agent with concrete citation material. The Analyst Agent generated complete papers consistently exceeding 4500

Received: 20-04-2026 | Accepted: 28-05-2026 | Published: 04-06-2026

International Journal of
AI EBIOMEDICINE INNOVATIONS
Peer Reviewed, Referred & Indexed Journal
ISSN: 3070-0507 www.zesterapublications.com Original Research Paper

words in total, with each of the nine sections meeting the minimum 400-word technical-depth requirement. The plain- prose formatting rules—no markdown bold, no bullet points inside sections, IEEE in-text citations of the form `[1]`, `[2]`, `[3]`—were respected throughout.

The real-time stage monitor, implemented via peri- odic `fetch` calls to the `/stage/` endpoint polling the `AGENT_STATE` dictionary, allowed users to observe pipeline progress live without a page reload. This transparency gave clear, moment-by-moment visibility into whether the Planner, Search, Analyst, or Reviewer stage was currently active, running, or complete, eliminating the perception of an opaque "black-box" process during the multi-minute synthesis.

B. AI Inference and Key-Rotation Resilience

A critical engineering outcome was the robustness of the `ask_ai` inference layer under the 15-requests-per-minute free-tier constraint of the Gemini API. The round-robin key-and-model rotation mechanism, which cycles across up to three API keys and three Gemini model variants (`gemini-2.5-flash`, `gemini-2.5-flash-lite`, `gemini-1.5-flash`) with a 65-second per-combo cooldown, prevented pipeline failures attributable to rate-limiting in all test runs where multiple keys were configured. When a `429/RESOURCE_EXHAUSTED` response was received, the system automatically marked the offending (key, model) pair, rotated to the next available combination, and resumed generation without user intervention. This design ensured a smooth research experience even under the constraints of the free tier.

C. Multilingual Output

The system was tested with the twelve language codes supported by the `LANGUAGE_NAMES` mapping in both the Planner and Analyst agents (English, Spanish, French, Ger- man, Hindi, Chinese, Japanese, Arabic, Portuguese, Russian, Italian, and Korean). In all cases, section titles, prose content, and in-text citations were produced in the requested language while the underlying IEEE structural conventions—section numbering, citation format, and reference list style—were pre- served. This demonstrates that the language parameterisation propagated correctly through every stage of the pipeline.

D. Export and Asset Generation

All three export pathways performed as intended during evaluation:

- **LaTeX Export:** The `/api/latex/` endpoint produced compilable `IEEEtran`-class LaTeX documents, complete with author blocks, abstract, IEEE keywords, numbered sections, equations, and a `thebibliography` refer- ence list. Documents were verified to compile without errors in Overleaf.
- **BibTeX Export:** The `/api/bibtex/` endpoint gener- ated eight to ten plausible IEEE-formatted `.bib` entries per paper, covering relevant conferences and journals from 2018–2024, ready for direct use with `cite` commands.
- **Clipboard Integration:** The client-side copy handler correctly transferred the full formatted report text to the system clipboard while preserving section headings and paragraph breaks.

Chart generation via `/api/chart/` produced Chart.js-compatible JSON structures with an AI-selected chart type appropriate to each section's content. Image generation via `/api/image/` successfully fell through the three-tier strategy—Imagen 3, Gemini 2.0 Flash experimental, and Pollinations.ai—in environments where a paid Gemini billing account was absent, ensuring that visual assets were always delivered to the user. The plagiarism checker returned orig-inality scores, verdicts, and explanatory notes for all tested inputs, with results persisted to the session dashboard for later review.

E. Frontend Performance and User Experience

The Django-template-based frontend rendered all four ap-plication pages (index, dashboard, result, settings) without measurable latency attributable to the template engine. The animated 3-D particle background rendered by `bg3d.js` op-erated independently of backend processing and introduced no perceptible degradation in UI responsiveness during pipeline execution. Asynchronous `fetch`-based updates to the result page allowed individual report sections to be refined via the AI-edit toolbar (improve, shorten, expand, simplify, make technical, translate) without triggering a full page reload, providing a fluid post-generation editing experience.

The dashboard aggregated all session-scoped outputs—research reports, chart and image assets, and plagiarism records—in a single view, giving users a clear record of their activity within the seven-day session window. The settings page exposed per-key, per-model rate-limit status in real time, enabling users to diagnose API availability and add additional Gemini keys at runtime without restarting the server.

F. Limitations and Observations

Several limitations were identified during evaluation. First, because `run_pipeline` is synchronous, the Django worker thread is blocked for the entire duration of report generation; under concurrent load this would exhaust the worker pool. Second, all session state is stored in module-level Python dictionaries, meaning that it is lost on server restart and is not shared across multiple worker processes. Third, the Reviewer Agent, while represented in `AGENT_STATE`, does not currently perform an independent quality-assessment pass; the `"reviewer"` state is set to `"complete"` immediately after the Analyst Agent returns. These points are addressed in the Future Work section.

Overall, the evaluation confirms that the Oretes Insight Sys-tem reliably automates the full research-report lifecycle—from topic ingestion through structured planning, web-grounded

Received: 20-04-2026 | Accepted: 28-05-2026 | Published: 04-06-2026

International Journal of
AI EBIOMEDICINE INNOVATIONS
Peer Reviewed, Referred & Indexed Journal
ISSN: 3070-0507 www.zesterapublications.com Original Research Paper

content synthesis, and multi-format export—delivering con- sistent, IEEE-compliant academic documents across diverse domains and languages.

VII. Future Enhancements

While the Oretes Insight System successfully automates the core research synthesis pipeline, the evaluation phase identified several concrete architectural limitations and user-experience gaps that motivate the following planned improvements.

Asynchronous Pipeline Execution: The current `run_pipeline` function in `views.py` is fully synchronous, blocking the Django worker thread for the entire duration of report generation. Under concurrent load this exhausts the worker pool and prevents simultaneous users from generating reports. A future version will replace this with Django Channels or Celery task queues, decoupling generation from the HTTP request–response cycle and enabling true concurrent multi-user operation. Real-time progress would then be pushed to the client via WebSockets rather than polled from the `/stage/` endpoint.

Persistent, Multi-Session State: All session data— research history, generated assets, and plagiarism records— is currently held in module-level Python dictionaries (`_research_history`, `_assets`, `_plag_checks`) that are lost on server restart and are not shared across worker pro- cesses. A future release will migrate this state to the existing SQLite database (or PostgreSQL for production deployments) using proper Django models, giving users durable access to their full report history across sessions and devices.

Fully Implemented Reviewer Agent: The `"reviewer"` entry in `AGENT_STATE` is currently set to `"complete"` im- mediately after the Analyst Agent returns, without performing an independent quality-assessment pass. A genuine Reviewer Agent will be introduced that evaluates each generated section against predefined thresholds for word count, citation density, and technical depth, and recursively resubmits failing sections to the Analyst Agent for refinement until all criteria are satisfied.

Live Academic Source Integration: The Search Agent currently queries the Tavily general web-search API, which does not guarantee peer-reviewed sources. Future versions will add direct integrations with arXiv, Semantic Scholar, and the IEEE Xplore API, enabling the system to retrieve and cite verified academic papers. This would replace synthesised BibTeX entries with real, verifiable references and improve the scholarly credibility of exported documents.

Multimodal Input Support: The Gemini models used in the inference layer (`ai_helper.py`) already support mul- timodal inputs natively. A planned enhancement will expose this capability through the frontend, allowing users to upload existing PDF papers, images of handwritten notes, or data tables that the Planner Agent can incorporate into the research outline and citation pool.

Advanced LaTeX Template Support: The `generate_latex` view currently produces only `IEEEtran`-class documents. Future iterations will extend the `/api/latex/` endpoint to accept a `template` parameter, supporting additional document classes such as `acmart` (ACM), `revtex4-2` (APS Physics), and `elsarticle` (Elsevier), making the system suitable for a wider range of publication venues.

Improved Image Generation Pipeline: Image generation currently relies on a three-tier fallback (Imagen 3 → Gem- ini 2.0 Flash → Pollinations.ai), with the third tier returning externally hosted URLs that may become unavailable. A future version will cache generated images as Base64 blobs in the database alongside their parent report, and will expose a dedicated asset library page so users can manage, reuse, and download visual assets independently of the report view.

Automated Presentation Export: Building on the existing `/api/latex/` and clipboard export pathways, a planned `/api/slides/` endpoint will transform the completed nine-section report into a structured slide deck (PowerPoint or Beamer LaTeX), mapping each section to a set of presentation slides with auto-extracted key points, section-level charts, and speaker notes derived from the full prose content.

Enhanced Plagiarism and Fact-Checking: The current `/api/plagiarism/` endpoint uses an LLM to estimate originality, which is a heuristic rather than a definitive cross-reference check. A future module will integrate with external originality APIs (e.g., Copyleaks or iThenticate) to perform genuine database-backed plagiarism detection, and will add a separate fact-checking pass that cross-references numerical claims and statistics against the retrieved source documents.

Per-User API Key Management: The runtime key- addition endpoint (`/api/add-key/`) currently appends keys to a settings list shared across all sessions. A future release will introduce proper user accounts with per-user Gemini API key vaults, usage-quota tracking, and key-health dashboards, separating individual key pools and enabling the system to scale to a multi-tenant production deployment.

VIII. Conclusion

The Oretes Insight System (Auto-Researcher) successfully demonstrates the design and implementation of an agent- driven platform that automates the end-to-end research work- flow. The system enables users to transform a single input topic into a comprehensive IEEE-formatted technical report exceeding 4500 words through the coordinated operation of multiple AI agents.

The application is implemented as a monolithic Django 5.1 web system, integrating SQLite for persistence and Google Gemini for large-language-model inference. The architecture ensures simplicity, reliability, and tight integration between backend processing and the user interface, while still sup- porting dynamic interaction through JavaScript-based asyn- chronous updates.

The multi-agent pipeline—comprising the Planner, Search, Analyst, and Reviewer stages—was shown to operate reliably across diverse domains and languages. The Planner generates structured research outlines, the Search Agent retrieves rele- vant web sources, and the Analyst produces detailed, IEEE-

Received: 20-04-2026 | Accepted: 28-05-2026 | Published: 04-06-2026

compliant academic content. The system further extends functionality through export modules (LaTeX and BibTeX), AI-powered chart and image generation, and plagiarism analysis.

The frontend, built using Django templates along with HTML, CSS, and JavaScript, provides a responsive and interactive user experience. Features such as real-time pipeline monitoring, session-based report tracking, and a dynamic 3-D animated background enhance usability and transparency.

Despite its effectiveness, the system has certain limitations. The synchronous pipeline execution can block server resources under heavy load, and the use of in-memory session storage limits scalability and persistence across server restarts. Additionally, the Reviewer stage currently does not perform an independent validation pass.

Overall, the Oretes Insight System provides a practical and efficient solution for automating technical report generation, significantly reducing the effort required for research drafting. Future improvements may include asynchronous task processing, database-backed session management, and enhanced validation mechanisms, further strengthening the system's scalability and robustness.

REFERENCES

[1] Django Software Foundation, "Django Documentation: The Web framework for perfectionists with deadlines," 2024. [Online]. Available: https://docs.djangoproject.com/
[2] SQLite Consortium, "SQLite Documentation: Lightweight relational database engine," 2024. [Online]. Available: https://www.sqlite.org/docs.html
[3] Google, "Generative AI (Gemini) Documentation: API for Gemini models," 2024. [Online]. Available: https://ai.google.dev/docs
[4] MDN Web Docs, "JavaScript Documentation: Client-side scripting for dynamic web applications," 2024. [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/JavaScript
[5] MDN Web Docs, "HTML and CSS Documentation: Structure and styling of web interfaces," 2024. [Online]. Available: https://developer.mozilla.org/
[6] LaTeX Project, "LaTeX Documentation: Typesetting system for technical and scientific documents," 2024. [Online]. Available: https://www.latex-project.org/help/documentation/
[7] Oren Patashnik, "BibTeX Documentation: Tool for formatting bibliographies," 2024. [Online]. Available: http://www.bibtex.org/
[8] IEEE, "IEEE Editorial Style Manual: Standards for technical report formatting and citation," 2024. [Online]. Available: https://ieeeauthorcenter.ieee.org/
[9] J. S. Park, J. C. O'Brien, C. J. Cai, M. R. Morris, P. Liang, and M. S. Bernstein, "Generative Agents: Interactive Simulacra of Human Behavior," *arXiv preprint arXiv:2304.03442*, 2023.
[10] P. Lewis, E. Perez, A. Piktus *et al.*, "Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks," *arXiv preprint arXiv:2005.11401*, 2020.
[11] J. Wei, X. Wang, D. Schuurmans *et al.*, "Chain-of-Thought Prompting Elicits Reasoning in Large Language Models," *arXiv preprint arXiv:2201.11903*, 2022.
[12] OpenAI, "Multi-Agent Systems and LLM-based Architectures: Concepts and Design Patterns," 2024. [Online]. Available: https://platform.openai.com/docs
[13] Turnitin, "Plagiarism Detection Techniques and Academic Integrity," 2024. [Online]. Available: https://www.turnitin.com/
[14] Overleaf, "LaTeX Documentation: Technical standards for PDF generation and export," 2024. [Online]. Available: https://www.overleaf.com/learn

Received: 20-04-2026 | Accepted: 28-05-2026 | Published: 04-06-2026